NAG Library Routine Document

D02HBF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D02HBF solves a two-point boundary value problem for a system of ordinary differential equations, using initial value techniques and Newton iteration; it generalizes subroutine D02HAF to include the case where parameters other than boundary values are to be determined.

2 Specification

```
SUBROUTINE D02HBF (P, N1, PE, E, N, SOLN, M1, FCN, BC, RANGE, W, SDW,          &
                   IFAIL)
INTEGER            N1, N, M1, SDW, IFAIL
REAL (KIND=nag_wp) P(N1), PE(N1), E(N), SOLN(N,M1), W(N,SDW)
EXTERNAL           FCN, BC, RANGE
```

3 Description

D02HBF solves a two-point boundary value problem by determining the unknown parameters $p_1, p_2, \ldots, p_{n_1}$ of the problem. These parameters may be, but need not be, boundary values; they may include eigenvalue parameters in the coefficients of the differential equations, length of the range of integration, etc. The notation and methods used are similar to those of D02HAF and you are advised to study this first. (The parameters $p_1, p_2, \ldots, p_{n_1}$ correspond precisely to the unknown boundary conditions in D02HAF.) It is assumed that we have a system of n first-order ordinary differential equations of the form:

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n), \qquad i = 1, 2, \ldots, n,$$

and that the derivatives f_i are evaluated by FCN. The system, including the boundary conditions given by BC and the range of integration given by RANGE, involves the n_1 unknown parameters $p_1, p_2, \ldots, p_{n_1}$ which are to be determined, and for which initial estimates must be supplied. The number of unknown parameters n_1 must not exceed the number of equations n. If $n_1 < n$, we assume that $(n - n_1)$ equations of the system are not involved in the matching process. These are usually referred to as 'driving equations'; they are independent of the parameters and of the solutions of the other n_1 equations. In numbering the equations for FCN, the driving equations must be put **first**.

The estimated values of the parameters are corrected by a form of Newton iteration. The Newton correction on each iteration is calculated using a Jacobian matrix whose (i, j)th element depends on the derivative of the ith component of the solution, y_i, with respect to the jth parameter, p_j. This matrix is calculated by a simple numerical differentiation technique which requires n_1 evaluations of the differential system.

If the parameter IFAIL is set appropriately, the routine automatically prints messages to inform you of the flow of the calculation. These messages are discussed in detail in Section 8.

D02HBF is a simplified version of D02SAF which is described in detail in Gladwell (1979).

4 References

Gladwell I (1979) The development of the boundary value codes in the ordinary differential equations chapter of the NAG Library *Codes for Boundary Value Problems in Ordinary Differential Equations.*

Lecture Notes in Computer Science (eds B Childs, M Scott, J W Daniel, E Denman and P Nelson) **76** Springer–Verlag

5 Parameters

You are strongly recommended to read Sections 3 and 8 in conjunction with this section.

1: P(N1) – REAL (KIND=nag_wp) array *Input/Output*

On entry: an estimate for the ith parameter, p_i, for $i = 1, 2, \ldots, n_1$.

On exit: the corrected value for the ith parameter, unless an error has occurred, when it contains the last calculated value of the parameter.

2: N1 – INTEGER *Input*

On entry: n_1, the number of parameters.

Constraint: $1 \leq \text{N1} \leq \text{N}$.

3: PE(N1) – REAL (KIND=nag_wp) array *Input*

On entry: the elements of PE must be given small positive values. The element PE(i) is used

(i) in the convergence test on the ith parameter in the Newton iteration, and

(ii) in perturbing the ith parameter when approximating the derivatives of the components of the solution with respect to this parameter for use in the Newton iteration.

The elements PE(i) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

Constraint: PE(i) > 0.0, for $i = 1, 2, \ldots, \text{N1}$.

4: E(N) – REAL (KIND=nag_wp) array *Input*

On entry: the elements of E must be given positive values. The element E(i) is used in the bound on the local error in the ith component of the solution y_i during integration.

The elements E(i) should not be chosen too small. They should usually be several orders of magnitude larger than ***machine precision***.

Constraint: E(i) > 0.0, for $i = 1, 2, \ldots, \text{N}$.

5: N – INTEGER *Input*

On entry: n, the total number of differential equations.

Constraint: $\text{N} \geq 2$.

6: SOLN(N,M1) – REAL (KIND=nag_wp) array *Output*

On exit: the solution when $\text{M1} > 1$.

7: M1 – INTEGER *Input*

On entry: a value which controls exit values.

$\text{M1} = 1$
The final solution is not calculated.

$\text{M1} > 1$
The final values of the solution at interval (length of range)/(M1 − 1) are calculated and stored sequentially in the array SOLN starting with the values of the solutions evaluated at the first end point (see RANGE) stored in the first column of SOLN.

Constraint: $\text{M1} \geq 1$.

8: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (i.e., the derivatives y_i'), for $i = 1, 2, \ldots, n$, at a general point x.

The specification of FCN is:

```
SUBROUTINE FCN (X, Y, F, P)

REAL (KIND=nag_wp) X, Y(*), F(*), P(*)
```

In the description of the parameters of D02HBF below, n and $n1$ denote the numerical values of N and N1 in the call of D02HBF.

1: X – REAL (KIND=nag_wp) *Input*

On entry: x, the value of the argument.

2: Y$(*)$ – REAL (KIND=nag_wp) array *Input*

On entry: y_i, for $i = 1, 2, \ldots, n$, the value of the argument.

3: F$(*)$ – REAL (KIND=nag_wp) array *Output*

On exit: the value of f_i, for $i = 1, 2, \ldots, n$. The f_i may depend upon the parameters p_j, for $j = 1, 2, \ldots, n_1$. If there are any driving equations (see Section 3) then these must be numbered first in the ordering of the components of F in FCN.

4: P$(*)$ – REAL (KIND=nag_wp) array *Input*

On entry: the current estimate of the parameter p_i, for $i = 1, 2, \ldots, n_1$.

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: BC – SUBROUTINE, supplied by the user. *External Procedure*

BC must place in G1 and G2 the boundary conditions at a and b respectively (see RANGE).

The specification of BC is:

```
SUBROUTINE BC (G1, G2, P)

REAL (KIND=nag_wp) G1(*), G2(*), P(*)
```

In the description of the parameters of D02HBF below, n and $n1$ denote the numerical values of N and N1 in the call of D02HBF.

1: G1$(*)$ – REAL (KIND=nag_wp) array *Output*

On exit: the value of $y_i(a)$, (where this may be a known value or a function of the parameters p_j, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n_1$).

2: G2$(*)$ – REAL (KIND=nag_wp) array *Output*

On exit: the value of $y_i(b)$, for $i = 1, 2, \ldots, n$, (where these may be known values or functions of the parameters p_j, for $j = 1, 2, \ldots, n_1$). If $n > n_1$, so that there are some driving equations, then the first $n - n_1$ values of G2 need not be set since they are never used.

3: P$(*)$ – REAL (KIND=nag_wp) array *Input*

On entry: an estimate of the parameter p_i, for $i = 1, 2, \ldots, n_1$.

BC must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: RANGE – SUBROUTINE, supplied by the user. *External Procedure*

RANGE must evaluate the boundary points a and b, each of which may depend on the parameters $p_1, p_2, \ldots, p_{n_1}$. The integrations in the shooting method are always from a to b.

The specification of RANGE is:

```
SUBROUTINE RANGE (A, B, P)

REAL (KIND=nag_wp) A, B, P(*)
```

In the description of the parameters of D02HBF below, $n1$ denotes the actual value of N1 in the call of D02HBF.

1: A – REAL (KIND=nag_wp) *Output*

On exit: a, one of the boundary points.

2: B – REAL (KIND=nag_wp) *Output*

On exit: the second boundary point, b. Note that B $>$ A forces the direction of integration to be that of increasing x. If A and B are interchanged the direction of integration is reversed.

3: P$(*)$ – REAL (KIND=nag_wp) array *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1, 2, \ldots, n_1$.

RANGE must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02HBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: W(N,SDW) – REAL (KIND=nag_wp) array *Output*

Used mainly for workspace.

On exit: with IFAIL $= 2$, 3, 4 or 5 (see Section 6), W$(i, 1)$, for $i = 1, 2, \ldots, n$, contains the solution at the point x when the error occurred. W$(1, 2)$ contains x.

12: SDW – INTEGER *Input*

On entry: the second dimension of the array W as declared in the (sub)program from which D02HBF is called.

Constraint: SDW $\geq$ 3N $+$ 14 $+$ max$(11,$ N$)$.

13: IFAIL – INTEGER *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Section 3.3 in the Essential Introduction).

On entry: IFAIL must be set to a value with the decimal expansion *cba*, where each of the decimal digits c, b and a must have a value of 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;

$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);

$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e., hard failure with all messages printed).

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

One or more of the parameters N, N1, M1, SDW, E or PE is incorrectly set.

IFAIL $= 2$

The step length for the integration became too short whilst calculating the residual (see Section 8).

IFAIL $= 3$

No initial step length could be chosen for the integration whilst calculating the residual.

Note: IFAIL $= 2$ or 3 can occur due to choosing too small a value for E or due to choosing the wrong direction of integration. Try varying E and interchanging a and b. These error exits can also occur for very poor initial choices of the parameters in the array P and, in extreme cases, because D02HBF cannot be used to solve the problem posed.

IFAIL $= 4$

As for IFAIL $= 2$ but the error occurred when calculating the Jacobian.

IFAIL $= 5$

As for IFAIL $= 3$ but the error occurred when calculating the Jacobian.

IFAIL $= 6$

The calculated Jacobian has an insignificant column. This can occur because a parameter p_i is incorrectly entered when posing the problem.

Note: IFAIL $= 4$, 5 or 6 usually indicate a badly scaled problem. You may vary the size of PE. Otherwise the use of the more general D02SAF which affords more control over the calculations is advised.

IFAIL $= 7$

The linear algebra routine used (F08KBF (DGESVD)) has failed. This error exit should not occur and can be avoided by changing the initial estimates p_i.

IFAIL $= 8$

The Newton iteration has failed to converge. This can indicate a poor initial choice of parameters p_i or a very difficult problem. Consider varying the elements PE(i) if the residuals are small in the monitoring output. If the residuals are large, try varying the initial parameters p_i.

IFAIL $= 9$
IFAIL $= 10$
IFAIL $= 11$
IFAIL $= 12$
IFAIL $= 13$

Indicates that a serious error has occurred in an internal call. Check all array subscripts and subroutine parameter lists in the call to D02HBF. Seek expert help.

7 Accuracy

If the process converges, the accuracy to which the unknown parameters are determined is usually close to that specified by you; the solution, if requested, may be determined to a required accuracy by varying E.

8 Further Comments

The time taken by D02HBF depends on the complexity of the system, and on the number of iterations required. In practice, integration of the differential equations is by far the most costly process involved.

Wherever they occur in the routine, the error parameters contained in the arrays E and PE are used in 'mixed' form; that is $\mathrm{E}(i)$ always occurs in expressions of the form

$$\mathrm{E}(i) \times (1 + |y_i|)$$

and $\mathrm{PE}(i)$ always occurs in expressions of the form

$$\mathrm{PE}(i) \times (1 + |p_i|).$$

Though not ideal for every application, it is expected that this mixture of absolute and relative error testing will be adequate for most purposes.

You may determine a suitable direction of integration a to b and suitable values for $\mathrm{E}(i)$ by integrations with D02PEF. The best direction of integration is usually the direction of decreasing solutions. You are strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation. You may select the channel numbers on which this output is to appear by calls of X04AAF (for error messages) or X04ABF (for monitoring information) – see Section 9 for an example. Otherwise the default channel numbers will be used, as specified in the Users' Note. The monitoring information produced at each iteration includes the current parameter values, the residuals and 2-norms: a basic norm and a current norm. At each iteration the aim is to find parameter values which make the current norm less than the basic norm. Both these norms should tend to zero as should the residuals. (They would all be zero if the exact parameters were used as input.) For more details, in particular about the other monitoring information printed, you are advised to consult the specification of D02SAF, and especially the description of the parameter MONIT there.

The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters p_i. If it seems that too much computing time is required and, in particular, if the values of the residuals printed by the monitoring routine are much larger than the expected values of the solution at b, then the coding of FCN, BC and RANGE should be checked for errors. If no errors can be found, an independent attempt should be made to improve the initial estimates for p_i.

The subroutine can be used to solve a very wide range of problems, for example:

(a) eigenvalue problems, including problems where the eigenvalue occurs in the boundary conditions;

(b) problems where the differential equations depend on some parameters which are to be determined so as to satisfy certain boundary conditions (see Example 2 in Section 9);

(c) problems where one of the end points of the range of integration is to be determined as the point where a variable y_i takes a particular value (see Example 2 in Section 9);

(d) singular problems and problems on infinite ranges of integration where the values of the solution at a or b or both are determined by a power series or an asymptotic expansion (or a more complicated expression) and where some of the coefficients in the expression are to be determined (see Example 1 in Section 9); and

(e) differential equations with certain terms defined by other independent (driving) differential equations.

9 Example

For this routine two examples are presented. There is a single example program for D02HBF, with a main program and the code to solve the two example problems given in Example 1 (EX1) and Example 2 (EX2).

Example 1 (EX1)

This example finds the solution of the differential equation

$$y'' = \left(y^3 - y'\right)/2x$$

on the range $0 \leq x \leq 16$, with boundary conditions $y(0) = 0.1$ and $y(16) = 1/6$. We cannot use the differential equation at $x = 0$ because it is singular, so we take a truncated power series expansion

$$y(x) = 1/10 + p_1 \times \sqrt{x}/10 + x/100$$

near the origin where p_1 is one of the parameters to be determined. We choose the interval as $[0.1, 16]$ and setting $p_2 = y'(16)$, we can determine all the boundary conditions. We take X1 $= 16$. We write $y = \mathrm{Y}(1)$, $y' = \mathrm{Y}(2)$, and estimate PARAM$(1) = 0.2$, PARAM$(2) = 0.0$. Note the call to X04ABF before the call to D02HBF.

Example 2 (EX2)

This example finds the gravitational constant p_1 and the range p_2 over which a projectile must be fired to hit the target with a given velocity.

The differential equations are

$$y' = \tan\phi$$
$$v' = \frac{-\left(p_1 \sin\phi + 0.00002v^2\right)}{v\cos\phi}$$
$$\phi\prime = \frac{-p_1}{v^2}$$

on the range $0 < x < p_2$, with boundary conditions

$$y = 0, \quad v = 500, \quad \phi = 0.5 \quad \text{at} \quad x = 0,$$
$$y = 0, \quad v = 450, \quad \phi = p_3 \quad \text{at} \quad x = p_2.$$

We write $y = \mathrm{Y}(1)$, $v = \mathrm{Y}(2)$, $\phi = \mathrm{Y}(3)$. We estimate $p_1 = \mathrm{PARAM}(1) = 32$, $p_2 = \mathrm{PARAM}(2) = 6000$ and $p_3 = \mathrm{PARAM}(3) = 0.54$ (though this last estimate is not important).

9.1 Program Text

```
!   D02HBF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module d02hbfe_mod

!     Data for D02HBF example programs

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter                   :: iset = 1, nin = 5, nout = 6
    Contains
      Subroutine fcn1(x,y,f,p)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(*)
        Real (Kind=nag_wp), Intent (In)      :: p(*), y(*)
!       .. Executable Statements ..
        f(1) = y(2)
        f(2) = (y(1)**3-y(2))/(2.0E0_nag_wp*x)
        Return
      End Subroutine fcn1
      Subroutine range1(a,b,p)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: a, b
```

```
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: p(*)
!       .. Executable Statements ..
        a = 0.1E0_nag_wp
        b = 16.0E0_nag_wp
        Return
      End Subroutine range1
      Subroutine bc1(g1,g2,p)

!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: g1(*), g2(*)
        Real (Kind=nag_wp), Intent (In)      :: p(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: z
!       .. Intrinsic Procedures ..
        Intrinsic                            :: sqrt
!       .. Executable Statements ..
        z = 0.1E0_nag_wp
        g1(1) = 0.1E0_nag_wp + p(1)*sqrt(z)*0.1E0_nag_wp + 0.01E0_nag_wp*z
        g1(2) = p(1)*0.05E0_nag_wp/sqrt(z) + 0.01E0_nag_wp
        g2(1) = 1.0E0_nag_wp/6.0E0_nag_wp
        g2(2) = p(2)
        Return
      End Subroutine bc1
      Subroutine fcn2(x,y,f,p)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: x
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: f(*)
        Real (Kind=nag_wp), Intent (In)      :: p(*), y(*)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: cos, tan
!       .. Executable Statements ..
        f(1) = tan(y(3))
        f(2) = -p(1)*tan(y(3))/y(2) - 0.00002E0_nag_wp*y(2)/cos(y(3))
        f(3) = -p(1)/y(2)**2
        Return
      End Subroutine fcn2
      Subroutine range2(a,b,p)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: a, b
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: p(*)
!       .. Executable Statements ..
        a = 0.0E0_nag_wp
        b = p(2)
        Return
      End Subroutine range2
      Subroutine bc2(g1,g2,p)

!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: g1(*), g2(*)
        Real (Kind=nag_wp), Intent (In)      :: p(*)
!       .. Executable Statements ..
        g1(1) = 0.0E0_nag_wp
        g1(2) = 500.0E0_nag_wp
        g1(3) = 0.5E0_nag_wp
        g2(1) = 0.0E0_nag_wp
        g2(2) = 450.0E0_nag_wp
        g2(3) = p(3)
        Return
      End Subroutine bc2
    End Module d02hbfe_mod
    Program d02hbfe

!     D02HBF Example Main Program

!     .. Use Statements ..
      Use d02hbfe_mod, Only: nout
```

```
!     .. Implicit None Statement ..
      Implicit None
!     .. Executable Statements ..
      Write (nout,*) 'D02HBF Example Program Results'

      Call ex1

      Call ex2

    Contains
      Subroutine ex1

!       .. Use Statements ..
        Use nag_library, Only: d02hbf, nag_wp, x04abf
        Use d02hbfe_mod, Only: bc1, fcn1, iset, nin, range1
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: h, x, x1, xh
        Integer                             :: i, ifail, m1, n, n1, outchn, sdw
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable     :: e(:), p(:), pe(:), soln(:,:),  &
                                               w(:,:)
!       .. Intrinsic Procedures ..
        Intrinsic                           :: real
!       .. Executable Statements ..
!       Skip heading in data file
        Read (nin,*)
!       m1: controls exit values,  n: number of differential equations,
!       n1: number of parameters.
        Read (nin,*) m1, n, n1
        sdw = 3*n + 14 + 11
        Allocate (e(n),p(n1),pe(n1),soln(n,m1),w(n,sdw))
        Write (nout,*)
        outchn = nout
        Write (nout,*)
        Call x04abf(iset,outchn)
!       p: estimates for the parameters p,  e: bound on the local error.
        Read (nin,*) p(1:n1)
        Read (nin,*) pe(1:n1)
        Read (nin,*) e(1:n)

        Write (nout,*) 'Case 1'
        Write (nout,*)
!       ifail: behaviour on error exit
!              =1 for quiet-soft exit
!       * Set ifail to 111 to obtain monitoring information *
        ifail = 1
        Call d02hbf(p,n1,pe,e,n,soln,m1,fcn1,bc1,range1,w,sdw,ifail)

        If (ifail==0) Then
          Write (nout,*) 'Final parameters'
          Write (nout,99999)(p(i),i=1,n1)
          Write (nout,*)
          Write (nout,*) 'Final solution'
          Write (nout,*) 'X-value     Components of solution'
          Call range1(x,x1,p)
          h = (x1-x)/real(m1-1,kind=nag_wp)
          xh = x
          Do i = 1, m1
            Write (nout,99998) xh, soln(1:n,i)
            xh = xh + h
          End Do
        Else
          Write (nout,99996) ifail
          If (ifail>1 .And. ifail<=5) Then
            Write (nout,99997) w(1,2), (w(i,1),i=1,n)
          End If
        End If

        Return

99999   Format (1X,1P,3E15.3)
```

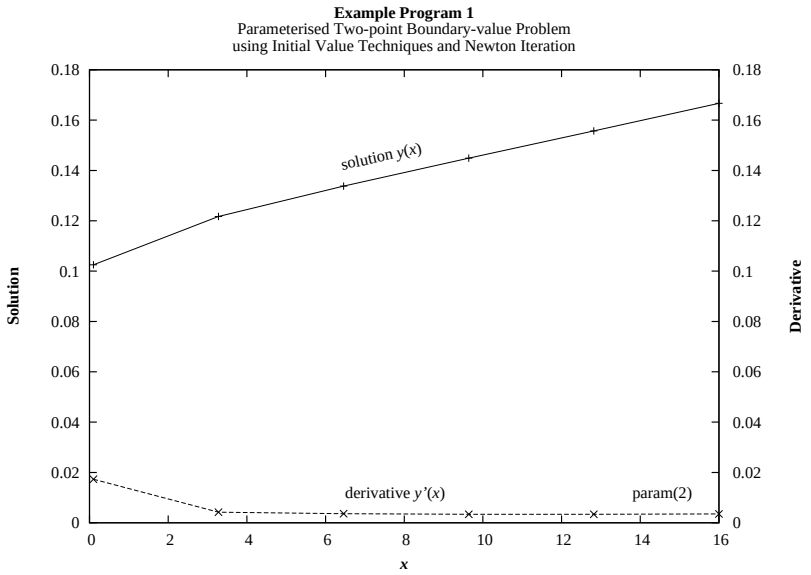
```
99998   Format (1X,F7.2,2F13.4)
99997   Format (/1X,'W(1,2) = ',F9.4,' W(.,1) = ',10E10.3)
99996   Format (1X/1X,' ** D02HBF returned with IFAIL = ',I5)
      End Subroutine ex1
      Subroutine ex2

!       .. Use Statements ..
        Use nag_library, Only: d02hbf, nag_wp, x04abf
        Use d02hbfe_mod, Only: bc2, fcn2, iset, nin, range2
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: h, x, x1, xh
        Integer                             :: i, ifail, m1, n, n1, outchn, sdw
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable     :: e(:), p(:), pe(:), soln(:,:),  &
                                               w(:,:)
!       .. Intrinsic Procedures ..
        Intrinsic                           :: real
!       .. Executable Statements ..
        Read (nin,*)
!       m1: controls exit values,  n: number of differential equations,
!       n1: number of parameters.
        Read (nin,*) m1, n, n1
        sdw = 3*n + 14 + 11
        Allocate (e(n),p(n1),pe(n1),soln(n,m1),w(n,sdw))
        outchn = nout
        Call x04abf(iset,outchn)
!       p: estimates for the parameters p,  e: bound on the local error.
        Read (nin,*) p(1:n1)
        Read (nin,*) pe(1:n1)
        Read (nin,*) e(1:n)

        Write (nout,*)
        Write (nout,*)
        Write (nout,*) 'Case 2'
        Write (nout,*)
!       ifail: behaviour on error exit
!              =1 for quiet-soft exit
!       * Set ifail to 111 to obtain monitoring information *
        ifail = 1
        Call d02hbf(p,n1,pe,e,n,soln,m1,fcn2,bc2,range2,w,sdw,ifail)

        If (ifail==0) Then
          Write (nout,*) 'Final parameters'
          Write (nout,99999)(p(i),i=1,n1)
          Write (nout,*)
          Write (nout,*) 'Final solution'
          Write (nout,*) 'X-value     Components of solution'
          Call range2(x,x1,p)
          h = (x1-x)/real(m1-1,kind=nag_wp)
          xh = x
          Do i = 1, m1
            Write (nout,99998) xh, soln(1:n,i)
            xh = xh + h
          End Do
        Else
          Write (nout,99996) ifail
          If (ifail>1 .And. ifail<=5) Then
            Write (nout,99997) w(1,2), (w(i,1),i=1,n)
          End If
        End If

        Return

99999   Format (1X,1P,3E15.3)
99998   Format (1X,F7.0,2F13.1,F13.3)
99997   Format (/1X,'W(1,2) = ',F9.4,' W(.,1) = ',10E10.3)
99996   Format (1X/1X,' ** D02HBF returned with IFAIL = ',I5)
      End Subroutine ex2
    End Program d02hbfe
```

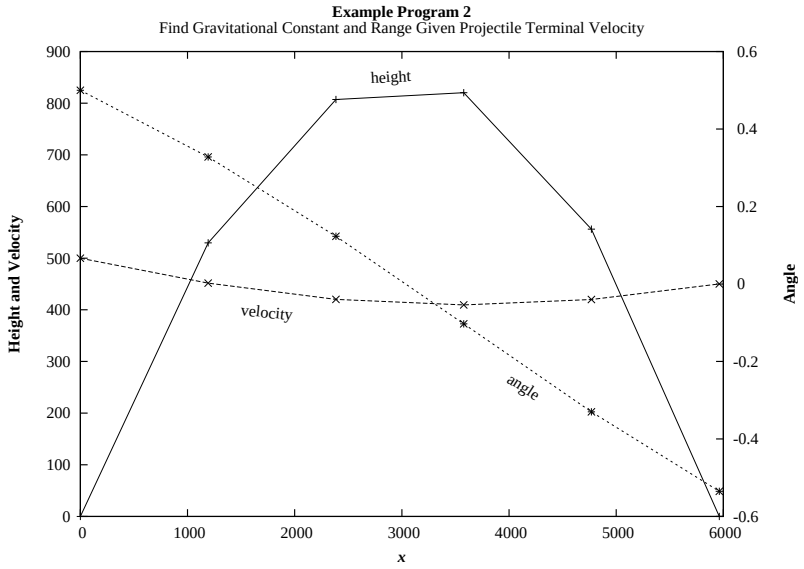
9.2 Program Data

```
D02HBF Example Program Data
   6 2 2                 : m1, n, n1
   0.2  0.0              : p
   1.0E-5 1.0E-3         : pe
   1.0E-4 1.0E-4         : e

   6 3 3                 : m1, n, n1
   32.0  6000.0  0.54    : p
   1.0E-5 1.0E-4 1.0E-4  : pe
   1.0E-2 1.0E-2 1.0E-2  : e
```

9.3 Program Results

```
 D02HBF Example Program Results


 Case 1

 Final parameters
       4.629E-02      3.494E-03

 Final solution
 X-value     Components of solution
    0.10       0.1025       0.0173
    3.28       0.1217       0.0042
    6.46       0.1338       0.0036
    9.64       0.1449       0.0034
   12.82       0.1557       0.0034
   16.00       0.1667       0.0035


 Case 2

 Final parameters
       3.239E+01      5.962E+03     -5.353E-01

 Final solution
 X-value     Components of solution
      0.          0.0        500.0         0.500
   1192.        529.6        451.6         0.328
   2385.        807.2        420.3         0.123
   3577.        820.4        409.4        -0.103
   4769.        556.1        420.0        -0.330
   5962.         -0.0        450.0        -0.535
```




Example Program 2
Find Gravitational Constant and Range Given Projectile Terminal Velocity
height
velocity
angle
Height and Velocity
Angle
x
900
800
700
600
500
400
300
200
100
0
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
1000
2000
3000
4000
5000
6000